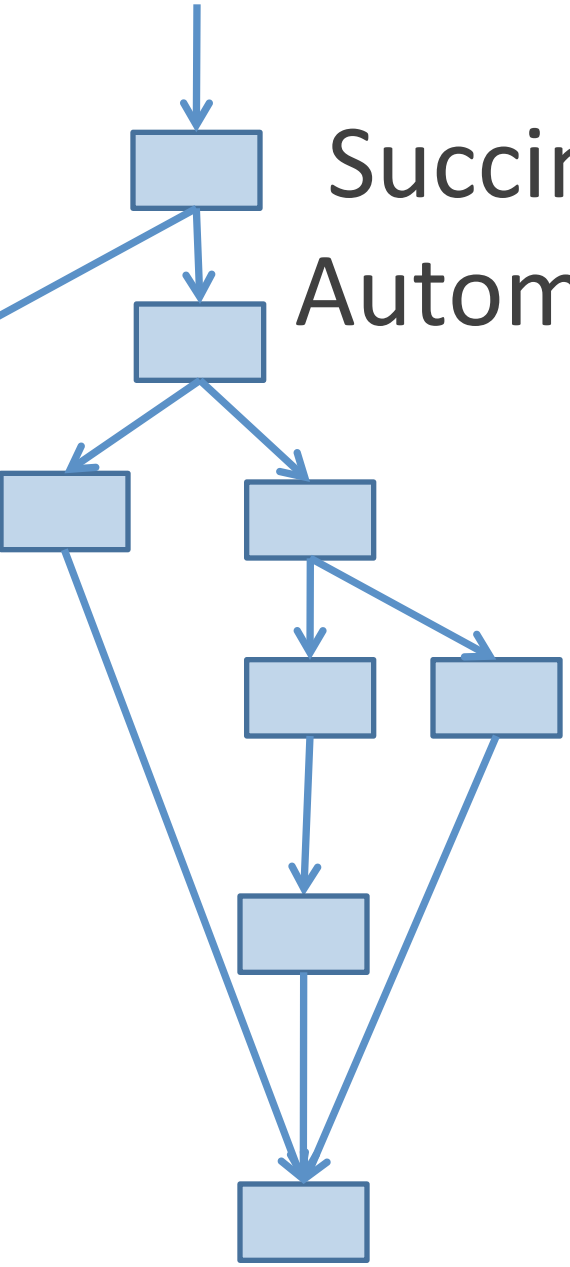


Succinct Analysis Specification with Automatic May/Must Determination

The DFAGEN Tool

Andrew Stone,
Michelle Strout,
Shweta Behere (Avaya)
Colorado State University

SCAM 2008 Conference
September 29th

Why Data-Flow Analysis?

For the M in SCAM: Optimization

For the A in SCAM: Program Understanding

Debugging

DF-Analysis is a common way or formulating flow- sensitive analyses

Why DF Analysis is Hard To Implement:

It's simple for a scalar only language

But these issues complicate things:

DFAGen: The DataFlow Analysis Generator

Solves issues on prior slide

Approach:

Automatic determination of may/must set usage

Requires:

A Control flow graph

Alias analysis results (with may/must information)

Side-effect analysis results

May or Must

```
int x, y, z;
int *p, *q;

x = rand() % 1000;
y = rand() % 1000;
z = rand() % 1000;

p = &x;
q = &y;

if(*p < *q) {
    q = &z;
}

*q = 500;
```

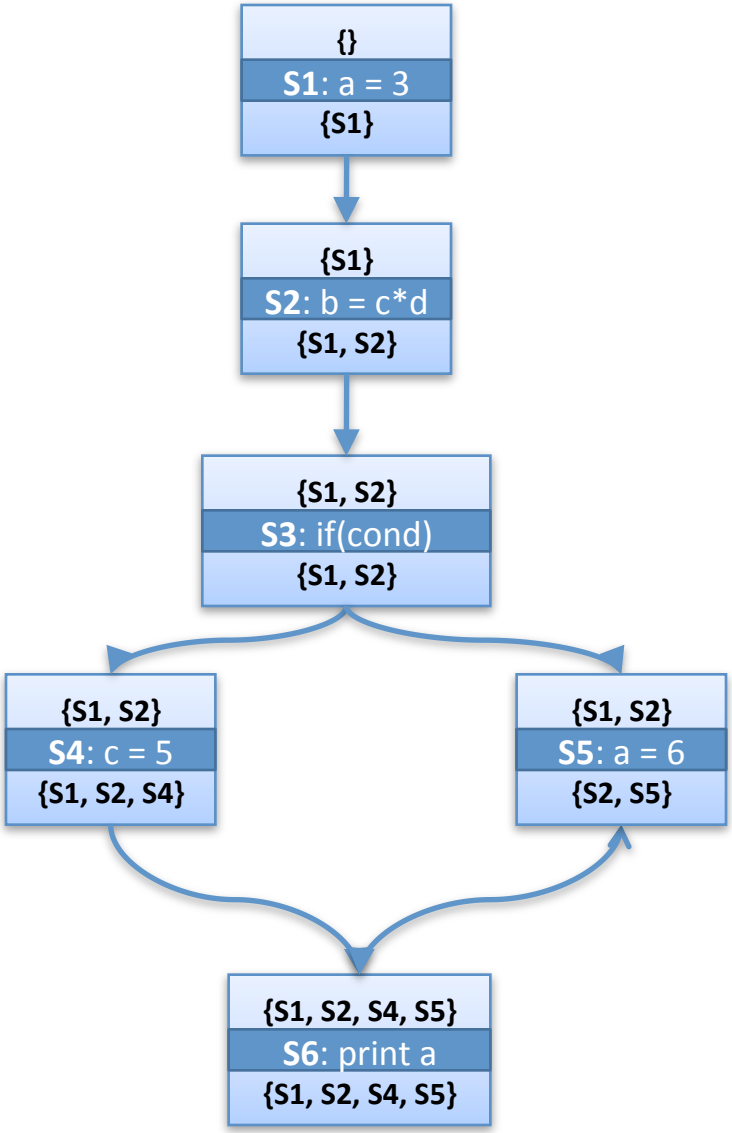
Must use: {x, y}

May use: {x, y}

Must def: {}

May def: {y, z}

Reaching Definitions Again



Meet: union

Direction: forward

Flowtype: statements

Style: may

GEN[s] = $\{s \mid defs[s] \neq \emptyset\}$

KILL[s] = $\{t \mid defs[t] \subseteq defs[s]\}$

reachingdefs.spec

```
analysis: ReachingDefs
  meet: union
  direction: forward
  flowtype: stmt
  style: may

  gen[s]:  { s | defs[s] !=empty }
  kill[s]: { t | defs[t] <= defs[s] }
```

So is it May or Must?

reachingdefs.spec

```
analysis: ReachingDefs
  meet: union
  direction: forward
  flowtype: stmt
  style: may

  gen[s]:  { s | defs[s] !=empty }
  kill[s]: { t | defs[t] <= defs[s] }
```

May

Must

How DFAGen Determines May/Must

Meet	Type	Gen	Kill
union	may	Upper	Lower
intersect	must	Lower	Upper

Operator	UB	UB	LB	LB
	lhs	rhs	lhs	rhs
<=	lower	upper	upper	lower
>=	upper	lower	lower	upper
<	lower	upper	upper	lower
>	upper	lower	lower	upper
union	upper	upper	lower	lower
intersect	upper	upper	lower	lower
!=empty	upper	-	lower	-

- Parse GEN and KILL expressions into an AST.
- Analyze in a top down fashion, determine whether we want an "upper bound" or "lower bound" value at each node.
- Use tables to left to determine these values.

More Detail

```
gen[s]:  { s | defs[s] !=empty }
kill[s]: { t | defs[t] <= defs[s] }
```

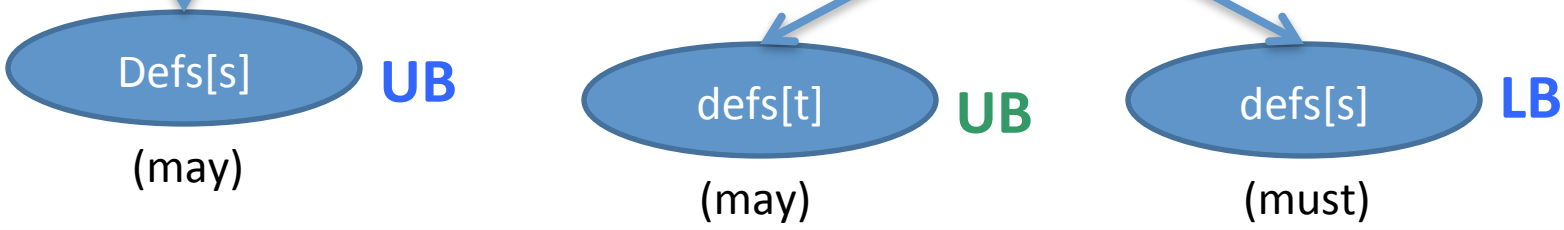
reachingdefs.spec

```
analysis: ReachingDefs
  meet: union
  direction: forward
  flowtype: stmt
  style: may

  gen[s]:  { s | defs[s] !=empty }
  kill[s]: { t | defs[t] <= defs[s] }
```

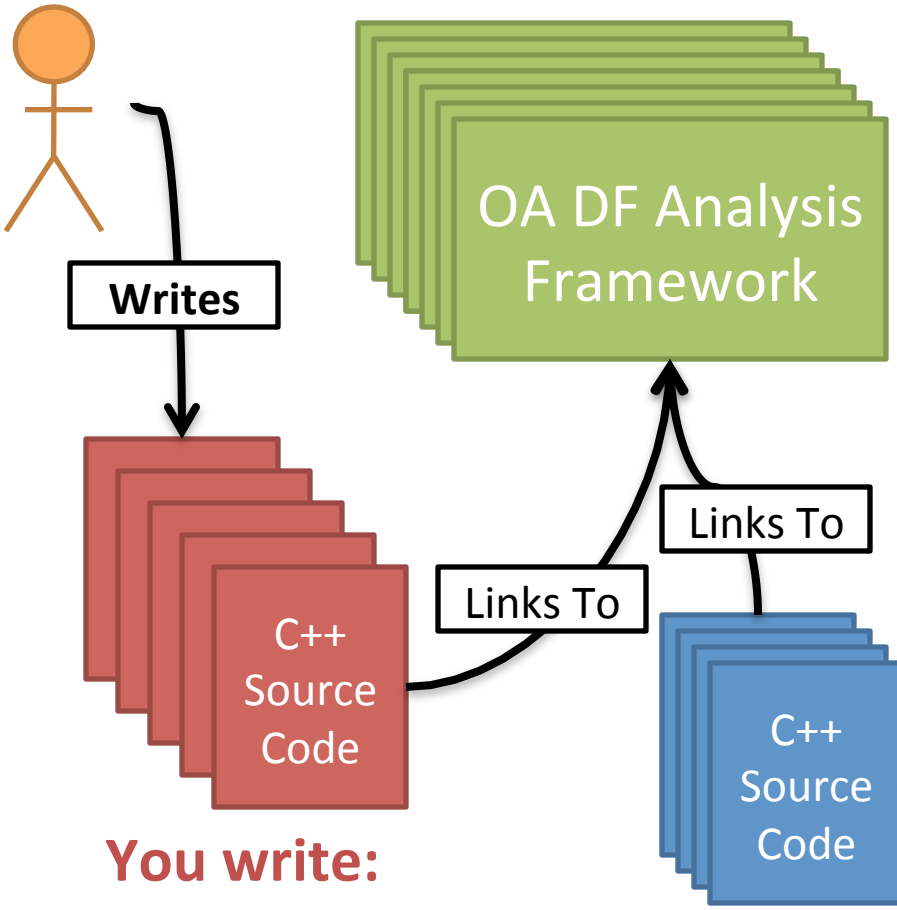
May

Must



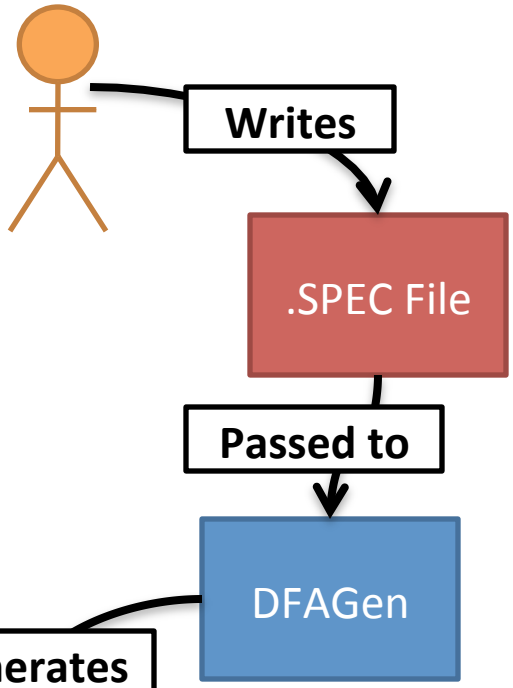
DFAGen Versus Hand-Written

Manual Version



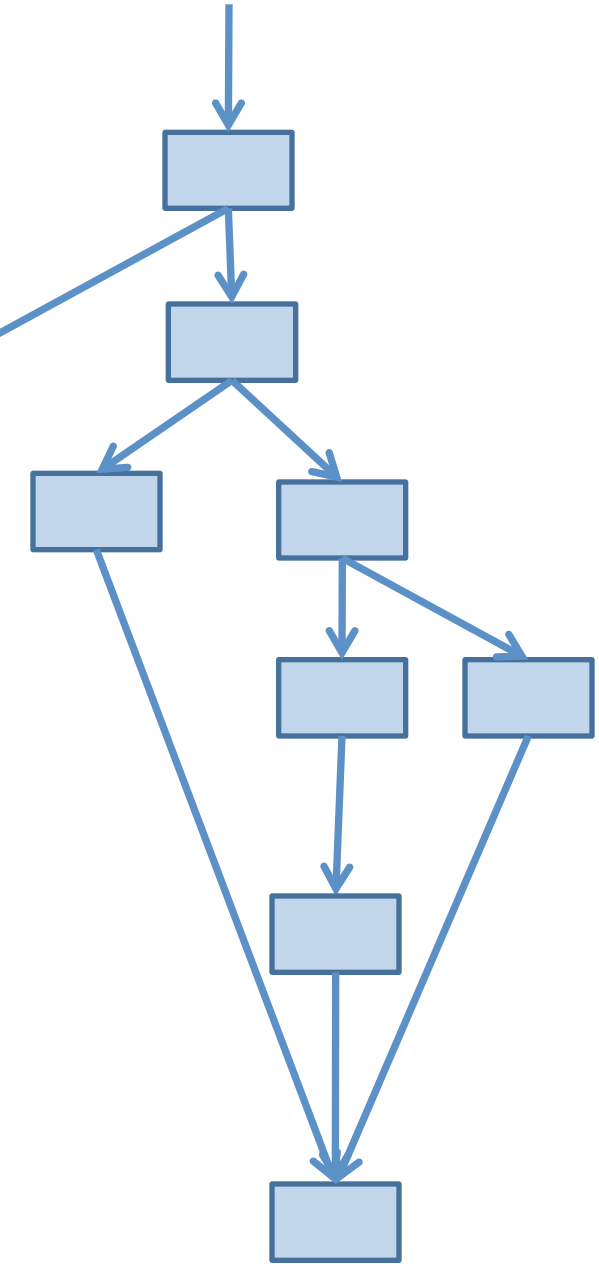
You write:
- **376 lines**

DFAGen Version



You write:
- **8 line SPEC file**

DFAGen Writes:
- **398 lines of C++ code**



Conclusions

DFAGen makes writing analyses easier.

It generates the analysis from succinct specifications.

DFAGen deals with the may/must issue of aliasing automatically.

We hope to prove that DFAGEN generated analyses are as fast a hand-written equivalents.